REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE 31 March, 1998	3. REPORT TYPE AND DATES COVERED Final Report: 1 Jul 96 - 31 Dec 97

4. TITLE AND SUBTITLE Search Strategies in Large-Scale Discrete Optimization: A Joint AI/OR Approach	5. FUNDING NUMBERS G: F49620-96-1-0335 PR: 2304/DS T: 2304/GS
6. AUTHOR(S) David W. Etherington David Joslin George L. Nemhauser	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) University of Oregon Office of Research Serv. and Admin. 5219 University of Oregon Eugene, OR 97403-5219 Georgia Institute of Technology School of Industrial and Systems Engineering Atlanta, GA 30332	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES) AFOSR/PKA 110 Duncan Avenue Suite B115 Bolling AFB, DC 20332-8080	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
11. SUPPLEMENTARY NOTES Program Manager: Dr. Abraham Waksman NM (202)767-5025	19980414 057
12a. DISTRIBUTION/AVAILABILITY STATEMENT Approved for public release; distribution is unlimited.	DTIC QUALITY INSPECTED 4

13. ABSTRACT *(Maximum 200 words)*

The aim of this award was to exploit and enhance the differing strengths of Artificial Intelligence (AI) and Operations Research (OR) in solving hard combinatorial optimization problems, discover synergies, and so develop better solution techniques.

By studying the strengths and weaknesses of the various approaches in the context of a large-scale manufacturing problem, a new AI solution approach capable of producing better solutions than traditional heuristic methods and handling larger problems than exact techniques was developed. This new approach generalizes a number of seemingly-divergent existing techniques and seems to be widely applicable.

In addition, it was discovered that OR techniques can be used to augment the new AI solver, resulting in significant improvements in both solution time and quality. This hybridized approach has also led to a new understanding of the OR technique known as "column generation," and these insights promise improvements in solution quality and time for a variety of OR problems.

14. SUBJECT TERMS			15. NUMBER OF PAGES 28
			16. PRICE CODE
17. SECURITY CLASSIFICATION OF REPORT UNCL	18. SECURITY CLASSIFICATION OF THIS PAGE UNCL	19. SECURITY CLASSIFICATION OF ABSTRACT UNCL	20. LIMITATION OF ABSTRACT UL

SEARCH STRATEGIES IN LARGE-SCALE DISCRETE OPTIMIZATION: A JOINT AI/OR APPROACH

Final Report

Grant number: F49620-96-1-0335

David W. Etherington
David Joslin

Computational Intelligence Research Laboratory
1269 University of Oregon
Eugene, OR 97403-1269
Email: {ether, joslin}@cirl.uoregon.edu

George L. Nemhauser

Georgia Institute of Technology
School of Industrial and Systems Engineering
Atlanta, GA 30332
Email: george.nemhauser@isye.gatech.edu

December 31, 1997

Contents

1 Executive summary

1.1 Technical progress

The fundamental question addressed by the research reported here is how integrating the different insights and methodologies of Artificial Intelligence (AI) and Operations Research (OR) can help us to search extremely large solution spaces. Both AI and OR search techniques have been applied to difficult combinatorial optimization problems, but their respective strengths and weaknesses are very different. OR algorithms tend to focus on finding optimal solutions, and as such must take a "global" view of a problem. In contrast, AI approaches are often based on local search, with a focus on finding good (but usually not optimal) solutions very quickly. Our research has been motivated by an effort to combine the best of both worlds.

We have made strong progress in three major areas. First, we have developed an innovative architecture, "Squeaky Wheel" Optimization (SWO), for AI local search. Second, we have developed a new hybrid AI/OR approach, H-OPT, that incorporates SWO and MINTO, a Linear Programming (LP) and Integer Programming (IP) solver. The hybrid approach outperforms both of the individual components. Third, we have developed some unorthodox, but highly effective, approaches to a traditional IP technique known as *column generation.* Each of these advances is discussed below.

Our development has been done using scheduling problems for a fiber-optic cable manufacturing plant, provided for us by Lucent Technologies. Using this domain allowed us to work on a well-defined, real-world problem for which real data could be obtained. The techniques that we have developed in this domain are quite general, and readily applicable to problems of more direct interest to the Air Force, such as target partitioning and assignment.

1.1.1 "Squeaky Wheel" Optimization

In looking at AI local search techniques for scheduling, we have been able to generalize several existing, highly effective scheduling algorithms, including Doubleback, previously developed at CIRL, and the patented algorithm used in OPTIFLEX, a commercial scheduler. The generalization is based on two principles that we have found to be key:

- Good solutions can be "taken apart" to reveal structure in the (local) search space.

- Local search benefits from the ability to make large, coherent moves in the search space.

Essentially, SWO generates a solution, then analyzes that solution to derive information about the local structure of the search space. That information then serves as feedback to guide the construction of a new solution. Because the new solution is constructed from scratch, the transition from one solution to the next is not limited to small, local moves as is the case with traditional local search algorithms. The new solution may differ from the previous solution at many points, but all of the changes are coherent in the sense that they are all motivated by the same feedback. The algorithm alternates between construction and analysis in this fashion until halted.

1.1.2 Hybrid AI/OR architecture

We have developed a hybrid architecture, H-OPT, that combines Integer Programming (IP) for global optimization, and AI local search techniques. Our hybrid approach captures the most desirable features of each. SWO is used to generate a large number of good feasible solutions quickly, and MINTO, an IP solver developed at Georgia Tech, is then used to combine the elements from those solutions into a better solution than the local search approach was able to find.

Experimental results are very encouraging. Traditional IP techniques alone were unable to generate feasible solutions to the largest problems in our test set, even when sub-optimal results were allowed. Although SWO is able to generate feasible (and quite good) solutions quickly, it fails to find some of the optimal solutions that the IP solver is able to find very quickly in H-OPT. Thus the hybrid approach outperforms both of its individual components.

1.1.3 New approaches to column generation

One approach to applying LP/IP techniques to very large problems is *column generation.* When the matrix defining the full problem is simply too large to generate, a relatively small number of columns may be created initially, where each column represents one way of solving a sub-problem. A solution over the initial set of columns is found, and feedback based on that solution is used to guide the generation of new columns. In this fashion, new columns are generated intelligently, and as a result, only a very small subset of all possible columns needs to be generated, keeping the problem to a manageable size.

Column generation has also traditionally been based on solving only isolated sub-problems, i.e., generating a new column without considering interactions between it and other columns. In this approach, the sub-problems are smaller than the overall problem, and it is possible to prove mathematically that with optimal solutions to the sub-problems, an optimal global solution will be found. Our experience, however, has been that this approach converges very slowly, and that part of the problem is that the columns thus generated do not "fit" well with other, existing columns. We have had substantially better success generating new columns by generating complete solutions, using SWO, so that the new columns are known to work well together, at least in that one solution. Columns generated in this fashion turn out to allow the IP solver to converge much more rapidly on good solutions which are better than any of the solutions generated by SWO alone.

Little or no attention has been paid to the importance of randomization in IP solvers that rely on column generation. Our experiments have demonstrated that controlled randomization can be effective at improving the performance of IP techniques. In applying randomization, we have focused on decisions for which we can generate a probability distribution that reflects the evaluation of each option by a heuristic function.

1.2 Personnel

At CIRL, at the University of Oregon, the research has involved David Etherington, David Joslin, and David Clements. Etherington and Joslin, the Principle Investigators, are faculty members, and Clements is a programmer.

Under the sub-contract with Georgia Tech, the research has involved the Principle Investigator, George Nemhauser, one other faculty member, Martin Savelsbergh, and a graduate student, Markus Puttlitz.

1.3 Publications and Patents

The following publications have acknowledged this grant:

- D. Clements, J. Crawford, D. Joslin, G. Nemhauser, M. Puttlitz and M. Savelsbergh, "Heuristic Optimization: A Hybrid AI/OR Approach."

Proceedings of the Workshop on Industrial Constraint-Directed Scheduling, 1997. (Held in conjunction with CP'97, Schloss Hagenberg, Austria.)

  This paper summarizes the major results for both SWO and H-OPT. It is anticipated that, with additional work extending the results to new domains, this will be turned into a journal paper.

- D. Joslin and D. Clements, "Squeaky Wheel Optimization." To appear in *Proceedings of the Fifteenth National Conference on Artificial Intelligence (AAAI-98)*, 1998.

  This paper describes the SWO framework in detail, and presents extensive experimental results.

- D. Joslin and A. Roy, "Exploiting Symmetries in Lifted CSPs." *Proceedings of the Thirteenth National Conference on Artificial Intelligence* (AAAI), 1997.

  This paper shows that it is possible to detect symmetries in problems by looking at problem descriptions that are "lifted," i.e., that incorporate quantification over finite sets. A similar approach would be readily adaptable to SWO, allowing it to avoid searching some redundant regions of the search space.

- C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, "Branch-and-Price: Column Generation for Solving Integer Programs." *Operations Research,* To appear in 1998.

  This paper discusses the fundamental concepts underlying of the use of column generation in the solution of integer programs.

A provisional patent application has been filed covering SWO and H-OPT, and we expect that a full application will be filed at the appropriate time. In addition, a publication on our new column generation techniques is in preparation, pending further experimental results.

1.4 Outline of the report

Section 2 describes the fiber optic cable manufacturing domain. Section 3 describes our "Squeaky Wheel" Optimization architecture. Section 4 describes our hybrid approach that combines SWO with MINTO, an IP solver. Section 5 outlines some of our most recent work, comparing traditional approaches to column generation with some unorthodox, but very effective, variations. The final section summarizes our conclusions, and discusses some of the directions for future research that we hope to pursue.

2 Experimental Framework

2.1 Experimental domain

This section describes the domain used for all of our experiments, and the LP/IP formulation that underlies both the H-OPT architecture (Section 4) and our new approaches to column generation (Section 5). This is a rather generic scheduling problem so the methodology developed should be applicable to a wide variety of scheduling problems and many other logistics problems.

The problem we have selected to work on is fairly representative, multi-job, parallel machine scheduling problem with lateness and changeover costs which originated in a fiber-optic cable plant.[1] A cable consists of up to 216 *optical fibers.* The *sheathing* operation involves joining the fibers and wrapping a protective rubber sheathing around them. This operation can be performed on one of 13 parallel *sheathing lines.* Typically, the number of cables in the set of orders is much larger than the number of sheathing lines. Every ordered cable has a release time and a due date. Production cannot begin before the release time, and the objective function includes a penalty for not completing a cable by the due date.

The production lines are heterogeneous in the types of cables they are able to produce, and the speeds at which they operate. For each cable, only a subset of the production lines will be compatible, and the time required to produce the cable will depend on which of the compatible lines is selected. Job preemption is not allowed, i.e. once a cable has started processing on a line, it finishes without interruption.

We need to make two types of decisions, namely how to assign cables, hereafter called jobs, to lines and how to sequence the jobs assigned to each line. Objectives are the minimization of the number of late jobs and the minimization of the sum of the setup times between jobs. This is an NP-hard combinatorial optimization problem.

2.2 LP/IP formulation

Our overall approach for both H-OPT and column generation is to formulate the problem as an IP problem, and to solve it by a branch-and-bound al-

[1]We wish to thank Robert Stubbs of Lucent Technologies for providing us with data to use for our experiments.

gorithm. Hence we need a "good" IP formulation, an efficient method for solving the linear programming (LP) relaxation, and an algorithm to generate integral solutions.

One concept of modeling discrete optimization problems with complicated constraints that has been shown to work well in practice is known as *set partitioning (SP)*. Suppose we assign *schedules* (rather than single jobs) to lines. Let a *line schedule* be a feasible assignment of a group of jobs to a line, including a sequencing and the associated objective cost. Notice that the computation of the objective function value of one line is independent of all other lines. To solve the overall problem, we need to find a min-cost subset of the set of all line schedules that uses each line at most once and includes each job in exactly one line schedule.

Let x_{lm} be the 0/1 decision variable which is 1 if line schedule l is assigned to line m. Associated with this variable will be a column a_{lm} representing:

- A set of jobs assigned to line m, represented by 0/1 indicators a^j_{lm}, which are equal to 1 if job j is in line schedule l and 0 otherwise. Column $a_{lm} = \{a^j_{lm}\}$ will then be the characteristic vector of the jobs in line schedule l for line m.

- Any ordering of that set of jobs results in a cost c_{lm} associated with that line schedule. For a given set of jobs, we would ideally like to find a line schedule that minimizes c_{lm}, but solving this problem is NP-hard, and in practice we usually must apply heuristic methods.

This leads to the SP problem

$$\begin{array}{lrcll}
\text{Minimize} & \sum_{m \in M} \sum_{l \in L_m} c_{lm} x_{lm} & & & \\
\text{subject to} & \sum_{m \in M} \sum_{l \in L_m} a^j_{lm} x_{lm} & = & 1 & \forall\, j \in J \\
& \sum_{l \in L_m} x_{lm} & \leq & 1 & \forall\, m \in M \\
& x_{lm} & \in & \{0,1\} & \forall\, l \in \bigcup L_m, m \in M
\end{array}$$

where L_m is the set of feasible line schedules for line m, J is the set of jobs, and M is the set of available production lines.

The SP formulation comprises two types of constraints. The first forces the solution to the scheduling problem to include each job exactly once. The second makes sure that for each line at most one line schedule can be part of the solution. Note that the constraints that determine whether or not a line schedule is feasible are not represented in the SP formulation, since only feasible line schedules are considered in this formulation. These feasible schedules are generated using the "squeaky wheel" optimization techniques to be discussed next.

Although fairly large instances of SP problems can be solved efficiently, the algorithmic challenge is to devise methods for solving SPs with a huge number of columns. In our scheduling problem, the SP has a column for every possible line schedule for every line. The number of such columns is generally exponential in the number of jobs. Fortunately, as explained in Section 5, it is possible to approximate the SP so that only a relatively small number of line schedules are considered.

3 "Squeaky Wheel" Optimization

This section describes "Squeaky Wheel" Optimization (SWO), an optimization architecture that also serves as a component in H-OPT (Section 4). SWO is a generalization of several existing, highly effective scheduling algorithms, including Doubleback Optimization [3] and the genetic algorithm used in OPTIFLEX, a commercial scheduler [12]. In SWO, solutions are analyzed to provide feedback for a local search algorithm. The algorithm is designed to make large "coherent" moves in the search space, thus helping to avoid local optima without relying entirely on random moves.

Some of the most effective approaches for solving systems of constraints in recent years have been based on local search. GSAT [11] and WSAT [10] apply local search techniques to solving propositional satisfiability problems, and WSAT has been used as the solver for the SATPLAN [8] planning system. CIRL's scheduling technology uses *Doubleback Optimization,* which performs a kind of local search to improve a "seed" schedule over a number of iterations [3]. The commercially successful scheduler OPTIFLEX from i2 Technologies is based on a patented approach that uses genetic algorithms [12]. Although these approaches differ substantially in the details, there has been a clear movement toward the use of local search in AI approaches to optimization problems.

In designing our local search algorithm, we began by looking at the Doubleback algorithm, because it had been extremely successful in solving a

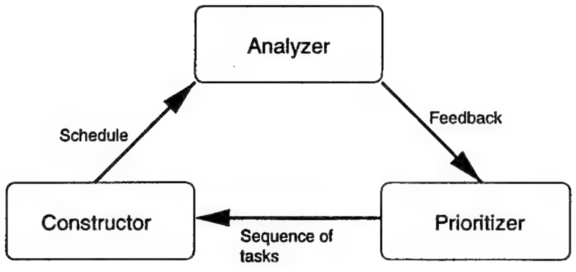

Figure 1: Local search architecture

standard type of scheduling problem. (On one benchmark related to aircraft manufacture [5], CIRL's scheduler produces the best-known solutions by a substantial margin, and finds them faster than the closest competitors.) However, the Doubleback algorithm is only useful when the objective is to minimize makespan (the time required to execute the schedule). Our problem domain required a different objective function, using a weighted sum of several factors. The problems also used constraints that are more complex than could be handled by the current Doubleback algorithm. Because of this, we began thinking about the principles behind Doubleback, looking for an effective generalization of that approach.

The architecture that emerged is summarized in Figure 1. There are three components:

Prioritizer Generates a sequence of jobs, with higher priority jobs being earlier in the sequence. Uses feedback from the Analyzer to modify previously generated sequences.

Constructor Given a sequence of jobs, constructs a schedule. Uses "greedy" scheduling for each job, in the order they occur in the sequence, without backtracking.

Analyzer Given a schedule, analyzes that schedule to find the "trouble spots." This feedback is provided to the Prioritizer.

We call this architecture "Squeaky Wheel" Optimization (SWO), from the aphorism "The squeaky wheel gets the grease." The idea is that on each iteration, the Analyzer determines which jobs are causing the most trouble in the current schedule, and the Prioritizer ensures that the Constructor gives more attention to those jobs in the next iteration. Note that this "priority" does not reflect the relative importance of a job, but rather the relative difficulty of finding a good place to put that job in a schedule.

In the current implementation, the Analyzer "assigns blame" to each of the jobs in the current schedule. For each job we calculate a simple lower bound on the minimum possible cost that each job could contribute to any schedule. For example, if a job has a release time that is later than its due date, then it will be late in every schedule, and the minimum possible cost already includes that penalty. The minimum possible setup costs are also included. Then, for a given a schedule, the penalty assigned to each job is its "excess cost," the difference between its actual cost and its minimum possible cost. The setup time penalty for each pair of adjacent jobs is shared between the two jobs, and the penalty for lateness is charged only to the late job itself.

Once these penalties have been assigned, the Prioritizer modifies the previous sequence of jobs by moving jobs with high penalties forward in the sequence. We currently move jobs forward in the sequence a distance that increases with the magnitude of the penalty, such that to move from the back of the sequence to the front, a job must have a high penalty over several iterations. (Sorting the jobs by their assigned penalty is simpler, and turns out to be almost as effective.) As a job moves forward in the sequence, its penalty will tend to decrease, and if it decreases sufficiently the job may then tend to drift back down the sequence as other jobs are moved ahead of it. If it sinks too far down the sequence, of course, its penalty may increase again, resulting in a forward move.

The Constructor builds a schedule by adding jobs one at a time, in the order they occur in the sequence. A job is added by selecting a line, and a position relative to the jobs already in that line. A job may be inserted between any two jobs already in the line, or at the beginning or end of that schedule, but changes to the relative positions of the jobs already in the line are not considered. Each job in the line is then assigned to its earliest possible start time, subject to the chosen ordering, i.e., a job starts at the minimum of either its release time or as soon as possible after the previous job on that line, allowing for the appropriate setup time between them.

For each of the possible insertion points in the schedule, relative to the jobs already in each line, the Constructor calculates the effect on the objective function, and the job is placed at the best-scoring location. Ties are broken randomly. After all of the jobs in the sequence have been placed, the Constructor tries to improve on the completed schedule with a small amount of local search. Currently, we only consider reordering jobs within a line.

The design of the local search architecture was influenced by two key insights:

- *Good solutions can reveal problem structure.* By analyzing a good solution, we can often identify elements of that solution that work well, and elements that work poorly. A resource that is used at full capacity, for example, may represent a bottleneck. This information about problem structure is local, in the sense that it may only apply to some part of the search space currently under examination, but it still may be extremely useful in helping figure out in what direction the search should go next.

- *Local search can benefit from the ability to make large, coherent moves.* It is well known that local search techniques tend to become trapped in local optima, from which it may take a large number of moves to escape. Random moves are a partial remedy, and in addition, most local search algorithms periodically just start over with a new random assignment. While random moves, small or large, are helpful, we believe our architecture works, in part, because of its ability to also make large *coherent* moves. A small change in the sequence of tasks generated by the Prioritizer may correspond to a large change in the corresponding schedule generated by the Constructor. Exchanging the positions of two tasks in the sequence given to the Constructor may change the positions of those two tasks in the schedule, and, in addition, allow some of the lower priority tasks, later in the sequence, to be shuffled around to accommodate those changes. This is a large move that is "coherent" in the sense that it is similar to what we might expect from moving the higher priority task, then propagating the effects of that change by moving lower priority tasks as well. This single move may correspond to a large number of moves for a search algorithm that only looks at local changes to the schedule, and may thus be difficult for such an algorithm to find.

This architecture can be thought of as searching two coupled spaces: the space of solutions, and the space of job sequences. Note that in the space of job sequences, the only "local optima" are those in which all jobs are assigned the same penalty, which in practice does not occur. Because of this, the architecture tends to avoid getting trapped in local optima in the solutions generated by the Constructor, since analysis and prioritization will always (in practice) suggest changes in the sequence, thus changing the solution generated on the next iteration. The randomization used in tie breaking will also tend to help avoid local optima.

Note that this architecture is a general framework, and not itself a specific algorithm. Doubleback can be viewed as an instance of this architecture, for example [3]. The OPTIFLEX scheduler [12] can also be viewed as an instance, with a genetic algorithm replacing the analysis phase. (In effect, the "analysis" instead emerges from the relative fitness of the members of the population.) These schedulers may appear to have little in common, but we believe that we have uncovered some principles that underlie both of these approaches. In the case of the OPTIFLEX scheduler, for example, we hypothesize that it is not genetic algorithms *per se* that make the scheduler so effective, but rather the manner in which prioritization and greedy construction are combined.

The importance of prioritization in greedy algorithms is not a new idea. The "First Fit" algorithm for bin packing, for example, relies on placing items into bins in decreasing order of size [6]. Another example would be GRASP (Greedy Randomized Adaptive Search Procedure) [4]. GRASP differs from our approach in several ways. First, the prioritization and construction aspects are more closely coupled in GRASP. After each element (here, a task) is added to the solution being constructed (here, a schedule), the remaining elements are re-evaluated by some heuristic. Thus the order in which elements are added to the solution may depend on previous decisions. Second, the order in which elements are selected in each trial is determined only by the heuristic (and randomization), so the trials are independent. GRASP has no mechanism analogous to the dynamic prioritization used in SWO, and consequently lacks the ability to search the space of sequences of elements, which we believe to be a key aspect of our architecture, because it allows the local search to make large, coherent moves.

Our architecture has been applied to both the single-line subproblem of generating new columns for an IP solver, and to the solution of the full scheduling problem. We currently focus on the use of this approach for solving the full problem; these solutions are also used to generate the initial set of columns for an LP/IP solver.

Our current implementation has considerable room for improvement. The analysis and feedback currently being used are very simple, and the construction of schedules could take various heuristics into account, such as preferring to place a job in a line that has more "slack," all other things being equal.

The experimental results for SWO are presented and discussed in Section 4.

4 Hybrid AI/OR architecture

4.1 Approach

Both heuristic and exact optimization techniques have been applied to difficult combinatorial optimization problems. Typically, heuristics have the advantage in speed and size of instances that can be handled, while exact methods have the advantage in quality. We present a hybrid approach that integrates heuristics and exact optimization techniques with the goal of capturing the desirable features of both.

We call our hybrid approach *heuristic optimization* (H-OPT). SWO, acting as one component of H-OPT, generates an initial set of good schedules, and the other component, MINTO, uses an Integer Programming algorithm to globally optimize those results, producing better schedules by combining elements of the schedules in the initial set.

In the optimization component, a linear program, which is a relaxation of an IP, is solved. Each column in the LP represents a feasible solution to a subproblem; in the problems used for the experiments for this paper, each column represents a feasible schedule for a single production line in a multi-line facility. Since there are a huge number of feasible schedules for each line, it is not practical to work with the whole LP. Instead, we use a local search heuristic to generate high-quality schedules.

A branch-and-bound solver is then used to obtain "good" integer solutions to the overall problem, i.e., finding the optimal combination of columns (line schedules) from the heuristically-generated schedules. Given a set of columns, the LP solver finds optimal primal and dual solutions to the LP relaxation. In future work, the optimal dual values will be used to guide a local search algorithm that will produce new columns for the LP throughout the search. In the scheduling problem, for example, this feedback indicates which jobs are most "difficult" to schedule.

SWO generates good solutions very quickly by itself, but in combination with the IP optimization, considerable improvements are obtained. The hybrid approach also produces better quality solutions than an existing TABU search algorithm, and runs faster.

In H-OPT, SWO generates as many good schedules as it can within a specified time limit. Each of these schedules contains one line schedule for each production line. Each individual line schedule becomes a column in the LP/IP formulation of a set partitioning problem, as previously discussed. A branch-and-bound solver is then used to find the optimal combination of columns. The solver used in our implementation is MINTO [9], a general purpose mixed integer optimizer that can be customized to exploit special problem structure through application functions. The LP relaxations are solved using CPLEX [2], a general purpose LP solver.

4.2 Experimental results

We have several sets of test data, ranging in size from 40 to 297 tasks (cable orders). In each problem there are 13 production lines. We compare the following solution methods:

TABU Uses TABU search [7], a local search algorithm in which moves that increase the cost are permitted to avoid getting trapped at a local optimum. To avoid cycling, when an "uphill" move is made, it is not allowed to be immediately undone.

SWO Applies the SWO architecture to the entire problem, running for a fixed number of iterations and returning the best schedule it finds.

H-OPT Uses the best schedules generated by SWO as the set of initial columns in the IP formulation described previously.

On the 297-task problem, our implementation of TABU was much less effective than either SWO or H-OPT, failing to find a feasible schedule after running for over 24 hours. On the smaller problems, TABU was able to find solutions, but both SWO and H-OPT outperformed TABU by a substantial margin.

Table 1 presents results for SWO and H-OPT on test sets with the number of jobs ranging from 40 to 297. In each case, ten trials were run and the results averaged. The second column of the table shows the best objective function value we have ever observed on each problem. The next two columns show

		SWO		H-OPT		
Data Set	Best Obj	Avg Obj	Avg Time	Avg Obj	Extra Time	% Impr.
40	1890	1890.0	162	1890.0	3	0.000
50	3101	3128.8	201	3127.2	3	0.051
60	2580	2580.6	242	2580.0	9	0.023
70	2713	2714.2	282	2713.0	5	0.044
148	8869	8951.7	604	8874.7	31	0.858
297	17503	17806.8	1209	17556.1	111	1.406

Table 1: Experimental results

the average objective function value, and the average time required for SWO. For H-OPT, the last three columns in the table show the average objective function value, the average time required by MINTO to optimize over the set of columns in the schedules generated by SWO, and the percentage improvement in the objective function. All times are in user processor seconds. These experiments were run on a Sparcstation 10 Model 50.

For the SWO/H-OPT experiments, we allowed the heuristic solver (SWO) to generate solutions up to a time limit proportional to the number of jobs, with approximately 20 minutes (1200 seconds) allowed for the largest problem (297 tasks). The line schedules from these solutions then formed the initial set of columns of H-OPT. H-OPT searches for a better combination of those columns. In other words, if it finds an improvement, it is the result of using columns from different schedules generated by SWO.

On the smallest problems in our test set, SWO by itself finds solutions that are as good as the best solutions we have found by any method. As the problem size increases, H-OPT is able to show greater improvements by recombining columns from the schedules found by SWO, with an improvement of 1.4% over the performance of SWO alone on the 297-job problem.

To further characterize the performance of H-OPT, we ran the largest data set (297 jobs) with varying amounts of time allowed for SWO to generate the "seed" schedules. Each column in Table 2 represents ten runs with a fixed amount of time allowed for seed generation. As shown in the table, MINTO required only a relatively small amount of time to improve upon the schedules produced by SWO. The degree of improvement ranged from 1.3% to 1.7%, corresponding to causing an additional 2 to 3 jobs to be completed by their

	Time allowed for SWO (seconds)					
	60	300	600	900	1200	1800
SWO	17979.4	17881.9	17818.1	17858.0	17806.8	17777.0
H-OPT	17715.0	17581.5	17553.8	17547.4	17556.1	17543.1
% impr.	1.460	1.678	1.482	1.738	1.406	1.315
Avg. MINTO time	11	49	125	121	111	110

Table 2: Experimental results (297 task problem)

due dates. (The actual improvement, of course, may be a combination of reducing lateness and reducing the total setup time.)

Figure 2 gives another view of the same data. For each of the six experiments shown in Table 2, the solid line shows the best schedule produced by SWO, on average, versus time. The dashed line segments show the results of taking the set of schedules generated by SWO up to some point (1 minute, 5 minutes, etc., up to 30 minutes) and allowing MINTO to optimize over that set of schedules. The horizontal line shows the best objective value we have ever observed, for reference.

As more time is allowed for SWO to search, better schedules are found, but with diminishing returns. On the other hand, until SWO has been allowed to run long enough to have produced a sufficient number of "good" schedules, MINTO does not have enough to work with. This suggests that H-OPT should take a dynamic approach to the boundary between SWO and MINTO. For example, SWO might be allowed to generate schedules, keeping a set of the best N schedules found, until Z consecutive iterations go by without any change to that set, for empirically determined N and Z. On the 297 task problem, H-OPT could have made the transition from SWO to MINTO as early as 300 seconds, with only a relatively small penalty.

Although the improvements achieved by MINTO are relatively small (on the order of 1.5%) MINTO achieves this improvement quickly, and SWO is unable to achieve the same degree of optimization even when given substantially more time. In cases where it is important to optimize as much as possible, and to do so quickly, the H-OPT combination of local search and global IP optimization can be highly effective. Although this is only a small experiment, we believe that these results clearly indicate that the approach is promising and should be explored further by both enhancing the techniques

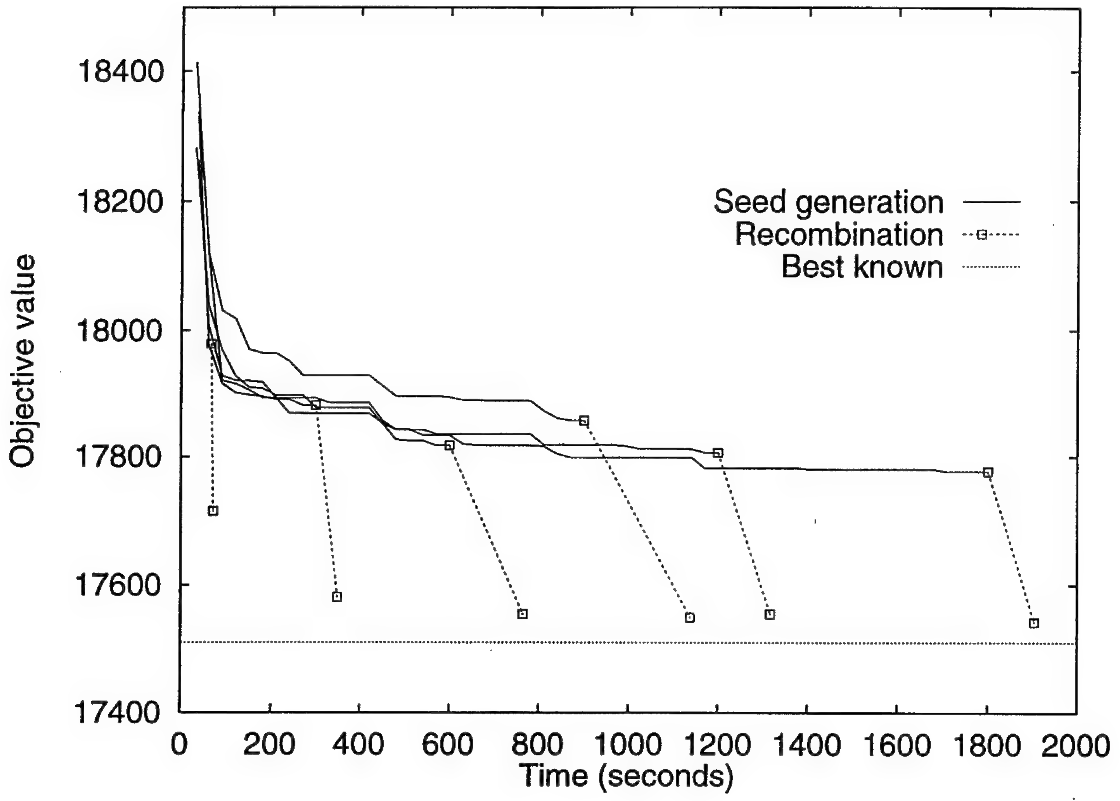

Figure 2: Time vs. objective function (297 task problem)

and the scope of applications.

5 New approaches to column generation

The OR community has a history of successfully solving large optimization problems that can be formulated as Linear Programming/Integer Programming (LP/IP) problems. For truly large problems it is often not computationally possible to pass a full definition of a problem (all possible rows and all possible columns) to an LP/IP engine. To even enumerate, let alone solve, such large problems would take more resources than can be bought.

However, truly large problems can often be decomposed and are therefore still amenable to LP/IP approaches. The particular type of problem we have been working on is such a problem. Two types of decision have to be made: assignment decisions that specify on which line a job will be executed, and sequencing decisions that specify the order in which jobs assigned to a line will be executed. Therefore, the problem can be decomposed into a master problem that focuses on the assignment decisions, and a subproblem that focuses on the sequencing decisions. More specifically, the master problem selects from among a set of available line schedules a minimum cost set of line schedules such that each job is executed exactly once, and the subproblem generates feasible line schedules with their associated costs.

The typical approach now starts with generating one or more initial complete schedules using heuristic methods. These initial schedules may range from poor to good in quality, but are usually not close to optimal. These initial heuristic schedules are used to provide an initial set of line schedules for the master problem.

The LP/IP engine then tries to select a good set of line schedules from among the initial set of line schedules. It recombines parts of the various initial solutions to provide the best LP solution that can be produced from those solutions. In doing so, it also identifies which jobs are giving the engine the hardest time. This information, i.e., the jobs that are making it hard to find a low cost complete solution, is then passed to the subproblem, which uses it to generate new line schedules. The generated line schedules attempt to address precisely those problems that were identified by the master problem. The newly generated line schedules are then fed back to the master problem and the process repeats with the LP/IP engine identifying new problem areas on each iteration.

The process of generating new line schedules is at the heart of this approach and is completely driven by LP considerations. Line schedules are generated so as to ensure that progress is made towards solving the LP relaxation of the master problem to optimality. Optimal LP solutions are necessary in the branch-and-bound paradigm to prove the optimality of the best known IP solution. In the context of heuristic optimization, where we want to find high quality solutions quickly, it may be more important to let IP considerations drive the generation of new lines schedules. The next section discusses preliminary experimental results suggesting that AI techniques can be very effective at column generation.

5.1 The idea of column generation

Although H-OPT currently uses MINTO only for the re-combination of columns generated by SWO, the H-OPT architecture also allows for introduction of new columns in response to the solutions produced by MINTO. We again use the LP/IP formulation of the set partitioning (SP) problem discussed in Section 2.

To solve the LP relaxation of the SP formulation, called the *master problem*, we use column generation, which means that the LP is solved with all of its rows, but not all of its columns present. The LP relaxation of SP including only a subset of the columns is called the *restricted master problem.* Columns are introduced into the restricted master problem in two phases. First, an initial set of columns is constructed to form the first restricted master problem. The LP relaxation of the restricted master is then solved to optimality. Second, new columns are incrementally added to the restricted master problem based on feedback from the LP solution. The same heuristic techniques that are currently used to generate the initial set of columns can be applied to the problem of generating additional columns. The problem of generating new columns using information from the current optimal LP solution to the restricted master problem is called the subproblem or pricing problem.

Given an optimal solution to the current LP relaxation of the restricted master problem, the *dual price* for each job j indicates how expensive it is for the current solution to cover job j. The idea of column generation is to use these dual prices to determine which jobs should be included in "good" new columns for the master problem. Since columns correspond to line schedules and there are different lines, the pricing problem consists of generating feasible schedules for different lines.

When solving pricing problems, we must evaluate or "price" candidate columns. Only candidate columns corresponding to feasible line schedules need to be evaluated, i.e. we require that all jobs included can run on that line. If we find a column whose cost c_{lm} is smaller than the sum of the dual prices of the jobs covered by that column, it is a candidate to enter the master problem. We say that such a column has a *negative reduced cost.* If no such column exists for any of the lines, the current solution to the LP relaxation of the restricted master problem is also the optimal solution to the LP relaxation of the entire master problem.

If we could solve the subproblems to optimality quickly, we would then have a fast LP solver which could then be embedded into a branch-and-bound algorithm for solving the entire scheduling problem. Unfortunately, the subproblems, while being easier than the original problem, are still NP-hard and a very large number of them need to be solved. Therefore we will solve the subproblems heuristically and stop generating columns when our heuristic terminates. This implies that we may not have solved the LP relaxation of the master problem to optimality and that the solution may not give a true lower bound. Thus, by using this approximate lower bound in the branch-and-bound phase of the algorithm we cannot guarantee that an optimal solution will be found to the entire scheduling problem.

5.2 Column generation based on IP vs. LP considerations

Some of the underlying principles of column generation approaches, derived from LP considerations, as they apply in our context, are

1. Try to generate columns with the most negative reduced costs.
2. Adding columns with positive reduced costs is not beneficial.
3. The pricing problem has to concentrate on generating line schedules in isolation from each other, since the master problem will combine line schedules into complete schedules.
4. The single line scheduling problem is easier to solve and therefore decomposing the problem reduces the overall complexity.

We discuss each assumption in turn.

Assumption 1:
Try to generate columns with the most negative reduced costs.

When generating new columns (either in isolation or using full solutions) each line schedule has a reduced cost associated with it. The reduced cost is the actual cost of the line schedule (in terms of time, materials, labor, etc.) plus the dual values for each job in the line schedule. A dual value indicates how much difficulty the most recent pass of the LP/IP engine had with a given job. Negative dual values indicate a difficult job for the engine to schedule; positive values indicate an easy job for the engine to schedule.

The magnitude of a dual value indicates how easy or how difficult the job was to schedule in the latest solution.

Linear programming theory tells us that giving the LP/IP engine new columns (line schedules in this case) that have a negative reduced cost will result in an improved solution to the LP relaxation of the master problem. If the column with maximum negative reduced cost has in fact a reduced cost greater than or equal to zero, then it has been proved that the LP solution is optimal also for the complete master problem.

However, pursuing the most negative reduced cost tends to create line schedules that will hardly ever be used in a final IP solution. Jobs that run on different lines in any good complete schedule are forced into each line schedule because they have very negative dual values. Therefore, from an IP point of view it may not be wise to restrict attention to the most negative reduced cost columns.

Assumption 2:

Adding columns with positive reduced costs is not beneficial.

It turns out that adding positive reduced cost columns can in fact move the LP/IP engine closer to a high quality complete schedule. There are 2 distinct cases:

1. A column with positive reduced cost is picked up by the LP/IP engine immediately after it is added and appears in the next LP/IP solution, improving it by some amount.

2. A column with positive reduced cost does not appear in the next LP/IP solution, but does appear in a subsequent LP/IP solution, improving it by some amount.

In the first case (immediate payoff case), the positive column is always added in conjunction with some other columns, at least one of which has a negative reduced cost. If the positive columns would not have been added at this time, then the LP/IP engine would have to go through one or more additional iterations to maybe add them later. This may or may not happen before running out of time, and even if it does the search may not succeed in reproducing those same columns.

A similar situation is happening when new positive reduced cost columns are not immediately useful, but are useful in the long run. In this case the search is effectively leaping ahead and producing columns that would have eventually had negative reduced cost anyway. Perhaps more important is the fact that columns important to solving the IP may only have negative reduced costs very late in the solution process and may not be added before we run out of time.

Assumption 3:

> The pricing problem concentrates on generating line schedules in isolation from each other, since the master problem will combine line schedules into complete schedules.

When the pricing problem considers the overall scheduling problem, the interactions between lines are handled much more effectively. Line schedules produced as part of a complete schedule tend to have groupings of jobs that are more efficient because the scheduler can see all of its options when deciding where to place jobs.

When generating complete schedules heuristically, it is unlikely that any generated schedule will be optimal. However, if the schedule produced is of high quality then it is likely that some parts of the complete schedule, i.e. some of the line schedules, will be optimal or near-optimal. Each generated schedule may have a different set of weak and strong points.

Assumption 4:

> The single line scheduling problem is easier to solve and therefore decomposing the problem reduces the overall complexity.

From a heuristic point of view, solving the line schedule problem is almost as hard as solving the complete schedule problem. We do not have to assign jobs to different lines, but we do have to select jobs that will run on the line currently under consideration. Because each line is considered in isolation from the others, it is very difficult to select the jobs that should be run on the line under consideration. As a result fairly uninformed choices are made.

When solving complete scheduling problems the solver has to deal with the increased complexity of the full problem, but it is able to make better-informed decisions because it can see the whole picture. When solving the complete scheduling problem, each decision takes more effort, but far, far fewer bad decisions are made. This tradeoff works in favor of generating complete schedules in every case we have seen.

Data set	Best objective	Standard columns	Full schedule	
		1890.9	1891.0	Seed objective
40	1890	1890.9	1890.0	Best objective
		7.3	27.9	Time
		3184.2	3189.6	Seed objective
50	3101	3180.0	3126.8	Best objective
		40.7	295.0	Time
		2602.1	2600.6	Seed objective
60	2580	2597.6	2580.0	Best objective
		47.6	292.2	Time
		2748.0	2742.9	Seed objective
70	2713	2747.1	2722.5	Best objective
		76.1	2808.0	Time
		9038.5	9038.0	Seed objective
148	8869	9036.1	8901.5	Best objective
		614.5	622.3	Time
		17892.8	17926.9	Seed objective
297	17503	17892.8	17617.8	Best objective
		652.7	1349.3	Time

Table 3: Column generation results

5.3 Experimental results

In order to evaluate our new approaches to column generation, we ran experiments comparing the following approaches:

1. "Standard" column generation, in which SWO is used to generate the most negative reduced cost columns it can find, with each column generated in isolation. This involves only a relatively minor change to SWO, essentially allowing it to solve a problem with a single production line, and without the restriction that all jobs must be scheduled somewhere.

2. Column generation by having SWO generate complete schedules, and entering all columns (with negative or positive reduced costs). Swo used dual values only to determine the initial priority sequence, and

otherwise only attempted to generate the best schedule possible, ignoring reduced costs.

In each experiment, a set of seed schedules provided the initial starting point (as with H-OPT), and column generation was allowed to proceed up to a time limit proportional to the size of the problem.

As the data in Table 3 show, generating full schedules is the most effective approach. In experiments currently under way, we are trying to further understand the tradeoffs involved in these approaches to column generation.

6 Summary and conclusions

Our results so far are very encouraging. As might be expected, based on other successful applications, our local search approach is capable of generating high-quality schedules very quickly. Allowing additional time does not improve those initial results dramatically. However, a combined exact and heuristic optimization approach allows a large number of good (and not so good) schedules to be "taken apart and recombined" in a way that quickly results in a higher quality schedule.

This hybrid approach takes advantage of the relative strengths of each part: local search is able to find good schedules, but tends to get stuck in local optima, and IP techniques provide a kind of global optimization that has no counterpart in local search. In a given solution, the local search approach may get some things right, and some things wrong, but the parts that are handled badly in one solution may be handled well in another. In a sense, global optimization allows the best parts of the different solutions to be combined.

The SWO architecture is itself innovative, and there is still a great deal of room for improvement. We are looking at more sophisticated methods of analysis and construction, and also looking at other domains that would require us to further generalize the approach taken here.

The use of randomness in integer programming algorithms has received little attention, and we believe the randomness introduced by SWO (in random tie-breaking during the construction of solutions) is partially responsible for the success of H-OPT. In future experiments, we hope to explore further the incorporation of randomization techniques. Within SWO, we can allow randomization to occur in any of the three main modules, and experiments are currently underway to try to understand the impact of randomization on the various parts of this architecture. In future work we will also experiment with a branch-and-price algorithm [1] in which randomized heuristics are used to generate new columns throughout the search tree.

Applying local search techniques for column generation may improve performance still further. The intuition behind this is that the dual values provided by the LP solver may provide valuable feedback to the local search engine about which tasks are "most critical." This can provide a bias toward different (and hopefully useful) areas of the search space. We have tried to use the dual values provided by MINTO to assist in the prioritization of jobs for SWO, but our results so far have been mixed. We hope in the future to better understand this mode of interaction between the two parts of H-OPT, and to show that the feedback that MINTO can provide can be very useful to SWO.

In the area of column generation, we have made some interesting discoveries that show that traditional LP-driven column generation may not be effective if the goal is to find a good solution quickly.

References

[1] C. Barnhart, E. Johnson, G. Nemhauser, M. Savelsbergh, and P. Vance. Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, 1996.

[2] CPLEX Optimization, Inc. Using the CPLEX callable library and CPLEX mixed integer library, version 4.0, 1996.

[3] J. M. Crawford. An approach to resource constrained project scheduling. In *Artificial Intelligence and Manufacturing Research Planning Workshop*, 1996.

[4] T. A. Feo and M. G. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[5] B. R. Fox and M. Ringer. Planning and scheduling benchmarks, 1995. `http://www.NeoSoft.com/ benchmrx/`.

[6] M. R. Garey and D. S. Johnson. *Computers and intractability: a guide to the theory of NP-completeness.* W. H. Freeman, 1979.

[7] F. Glover and M. Laguna. *Tabu Search.* Kluwer, 1997.

[8] H. Kautz and B. Selman. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proceedings of the National Conference on Artificial Intelligence*, Portland, OR, 1996.

[9] G. Nemhauser, M. Savelsbergh, and G. Sigismondi. Minto, a mixed integer optimizer. *Operations Research Letters*, 15:47–58, 1994.

[10] B. Selman, H. A. Kautz, and B. Cohen. Local search strategies for satisfiability testing. In *Second DIMACS challenge workshop on cliques, coloring, and satisfiability*, Rutgers University, October 1993.

[11] B. Selman, H. Levesque, and D. Mitchell. A new method for solving hard satisfiability problems. In *Proceedings of the National Conference on Artificial Intelligence*, pages 440–446, 1992.

[12] G. P. Syswerda. Generation of schedules using a genetic procedure, 1994. U.S. Patent number 5,319,781.